Model Checking

UE Simulation

Hélène Coullon

Table of contents

Introduction

Pourquoi modéliser un système ?

Expliquer de façon précise les objectifs et le fonctionnement d'un système

Modélisation textuelle

Expliquer le fonctionnement d'un système en langage naturel

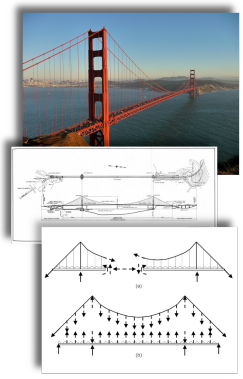
- Avantages : facile à lire et comprendre par un humain
- Désavantages : peu fiable et peut être compris de différentes façons

Modélisation formelle

Modéliser “formellement” (mathématiquement) le système

- Avantages : fiable et précis, interprétable par des outils automatiques
- Désavantages : difficile à apprendre et lire pour un humain

La modélisation en physique ?



Exemple de système à étudier : un pont

- Modélisations informelles : maquette à échelle plus petite
- Modélisation formelle : modèle équationnel

Vérifier un système ?

S'assurer que le système

- se comporte correctement (comme attendu),
- est fiable,
- est sûr,
- détecter des problèmes dans le système et les résoudre.

Tester un système

Principes

- éllaboration de tests pour vérifier le bon comportement
- basé sur la pertinence des tests et l'observation

Approche traditionnelle pour vérifier un système

- Avantages :
  - relativement facile à mettre en oeuvre
  - utile à différentes étapes de la conception
- Désavantages :
  - exhaustivité des tests est souvent impossible
  - l'indéterminisme augmente la difficulté des tests (hasard, parallélisme etc.)

> "Tester des programmes peut être un moyen très efficace de révéler des bugs, mais est irrémédiablement inadapté pour en démontrer l'absence." [E.W. Dijkstra]

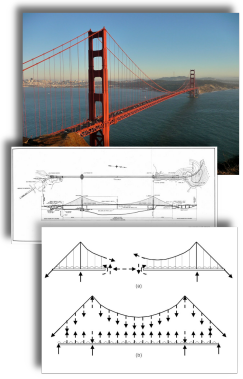
Vérification formelle

Principes

- représentation abstraite et formelle des propriétés que doit vérifier le système
- techniques de raisonnement mathématiques sur le modèle pour en vérifier des propriétés

- Avantages :
  - non ambigu
  - fiabilité
- Désavantages : difficile à mettre en oeuvre

En physique ou en maths ?



Exemple de système à étudier : un pont

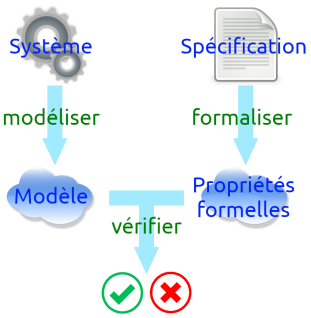
Exemple de propriété à étudier : la résistence au vent

- Test : maquette en soufflerie
- Vérification formelle : modèles équationnels

Méthodes formelles

C'est l'ensemble des outils mathématiques dont le but est de démontrer la validité d'un système ou d'un programme par rapport à une certaine spécification.

Elles permettent de raisonner rigoureusement, à l'aide de logique mathématique



Et dans ce module ?

Il existe beaucoup de méthodes formelles, basées sur des logiques différentes de raisonnement automatique.

On s'intéresse ici aux techniques adaptées aux systèmes réactifs et concurrents

- Système informatiques (processus de calcul et mémoire partagée, cloud etc.)
- Systèmes et procesus industriels (chaîne de montage, logistique etc.)
- Voitures autonomes
- etc.

Méthodes formelles adaptées à ces systèmes

- modélisation formelle : automates, réseaux de Petri
- vérification formelle : model-checking
- logique sous jacente utilisée : logique temporelle

Principe du model-checking

Le model-checker prend en entrée

- la modélisation d'un système (réseau de Petri, automates etc.)
- une propriété temporelle

Le model-checker retourne en sortie

- si la propriété est vrai ou fausse
- un contre-exemple si la propriété est fausse

Description du module

Informations générales

- 2 séances de 3h45
- 1 quizz individuel en début de séance 2
- Cours
- TD évalué
- Projet progressif en séance évalué

Contact : helene.coullon@imt-atlantique.fr

Bureau : B220

Compétences

- CG1 Comprendre et analyser, synthétiser un problème et/ou une situation complexe (TD, quizz et projet)
- CG4 Critiquer et décider (projet)
- CG14 S'engager (évaluation dans les séances)

Logique temporelle

Les logiques temporelles

Une logique temporelle s'intéresse à l'aspect temporel d'une propriété :

- Le langage pour décrire les contraintes/propriétés du système à un instant donné
  - "j'aime le chocolat"
  - "le robot ne dévisse pas ses propres boulons"
  - "le programme ne supprime pas l'ensemble de mes données"
  - "la place P contient n jetons"
- L'ajout de modalités temporelles (CTL par exemple)
  - "il est possible que"
  - "il est certain que"
  - "toujours"
  - "à un moment" etc.

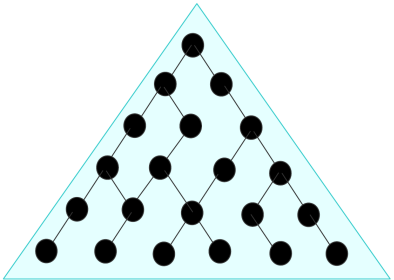
CTL

Il existe plusieurs types de logique temporelles : LTL, CTL, CTL*

Ici nous étudions CTL

- Computational Tree Logic (CTL)
- permet d'exprimer des propriétés formelles temporelles sur un réseau de Petri notamment

Principes



La structure de Kripke utilisée en CTL est un arbre (fini) où les noeuds représentent

- l'état actuel du système : la racine
- les états futurs : les noeuds enfants

Dans le cas d'un réseau de Petri chaque noeud est un marquage accessible depuis le marquage parent !

Fonctionnement d'un model-checker

Le model-checker

- construit la structure de Kripke à partir de la modélisation en réseau de Petri/automate donné en entrée (le système à étudier),
- applique des algorithmes de parcours de graphe optimisés pour vérifier une propriété temporelle sur le système,
- effectue une vérification exhaustive (formelle)

Attention la complexité du model-checking est exponentielle, de très gros systèmes ne peuvent être vérifiés sans appliquer des techniques avancées de réduction de l'espace de recherche (non vues dans ce module)

Connecteurs et quantificateurs CTL

Les connecteurs temporels de la propriété considèrent un chemin donné dans l'arbre et expriment les faits suivants

- **F** = Finally
  - la propriété doit être satisfaite par au moins un état dans le chemin
- **G** = Globally
  - la propriété doit être satisfaite par tous les états du chemin
- **X** = neXt
  - la propriété doit être satisfaite dans l'état suivant de l'état courant
- **U** = Until
  - la première propriété doit être satisfaite par tous les états du chemin jusqu'à ce que la deuxième propriété devienne vraie à un moment

et les quantificateurs de chemins définissent les chemins à considérer

- **A** = Along all paths
- **E** = There Exists a path

Formule CTL

Une formule (propriété) CTL combine

1. les quantificateurs
2. les connecteurs
3. la propriété / les contraintes à respecter

Exemple

`AG(‘‘j’aime le chocolat’’)` =

J'aime le chocolat maintenant et pour toujours.

Formules possibles 1/2

Pour φ (ψ) une propriété et un arbre de Kripke de mon système

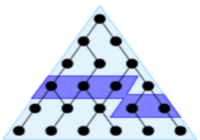
AF_φ	pour tous les chemins	φ est finalement satisfait à un moment
AG_φ	pour tous les chemins	tous les états satisfont φ
AX_φ	pour tous les chemins	φ est satisfait à l'état suivant
$A\lceil\varphi U\psi\rceil$	pour tous les chemins	il y a un état qui satisfait ψ et tous les états avant satisfont φ

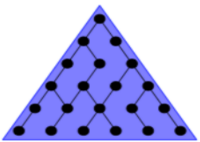
Formules possibles 2/2

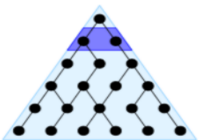
Pour φ (ψ) une propriété et un arbre de Kripke de mon système

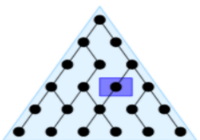
EF_φ	pour au moins 1 chemin	φ est finalement satisfait à un moment
EG_φ	pour au moins 1 chemin	tous les états satisfont φ
EX_φ	pour au moins 1 chemin	φ est satisfait à l'état suivant
$E[\varphi U \psi]$	pour au moins 1 chemin	il y a un état qui satisfait ψ et tous les états avant satisfont φ

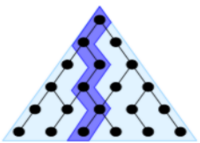
Visuel des formules CTL

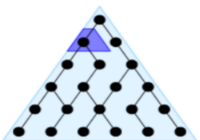


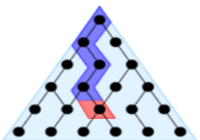












Les grands types de formules/propriétés

Sûreté (Safety)

- Le système n'entre jamais dans un état non sûr (non fiable, non souhaité)
- formules de type $AG(\neg\varphi)$ ou $EG(\neg\varphi)$

Accessibilité

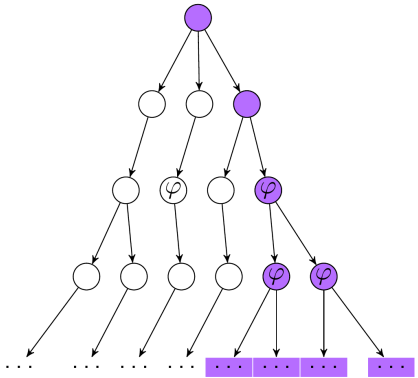
- Le système finit toujours par atteindre un état qui vérifie φ
- formules de type $AF(\varphi)$ ou $EF(\varphi)$

Vivacité (Liveness)

- Si une précondition p est satisfaite alors on finira par atteindre un état qui satisfait φ
- Il n'y a pas de blocage dans l'avancement
- Propriétés de type $EG(p \rightarrow AF\varphi)$ ou $AG((p \rightarrow AF\varphi)$

Exemple d'imbrications plus complexes

$EF(AG(\varphi))$ = il existe un chemin où au bout d'un moment sur tous les sous-chemins possibles φ est toujours vrai



Dans ce module : Pas de X, pas d'imbrications, il est probable que vous n'utilisez pas E aussi

TD propriétés temporelles

TD par 2 sur les propriétés temporelles

Propriétés à un instant donné

Et les contraintes sur la marquage alors ?

Generalized Mutual Exclusion Constraints (GMEC)

- Identifiant / nom des places
- le marquage d'une place donnée `M(P)`
  - retourne le nombre de jeton dans la place à l'instant t
- opérateurs numériques, égalité et innégalités
  - `+,-,*`
  - `<, <=, >, >=, ==, !=`
- opérateurs booléens
  - `and, or, =>, not`
- d'autres éléments non utilisés ici

Exemples GMEC et exemples complets

Exemples

- `M(P1)==1` : La place $P1$ contient un jeton
- `M(P2)-M(P3)>0` : Le nombre de jetons dans $P2$ moins le nombre de jetons dans $P3$ doit être supérieur à 0
- `(M(P3)==1 and M(P4)==1) or M(P2)==1` : il y a un jeton dans $P3$ et $P4$, ou bien un jeton dans $P2$

Exemples CTL complets

- `AG(M(P1)==1)` : Sur tous les chemins, le système a toujours un unique jeton dans $P1$
- `A(M(P2)==1) U (M(P3)>2)` : Sur tous les chemins, le système a un jeton dans $P2$ jusqu'à ce qu'il ait plus de 2 jetons dans $P3$
- `AG(M(P2)==1 => AF(M(P5)==1))` : Sur tous les chemins, si $P2$ a un jeton alors $P5$ finira par avoir un jeton

Roméo

Dans ce module nous allons utiliser le model-checker Roméo (chercheurs à Centrale Nantes)

- modélisation de son système en réseau de Petri
- contraintes avec GMEC
- logique (T)CTL (temporisé) allégé
  - Pas de **X** (neXt)
  - Pas de formules imbriquées sauf le `AG( X => AF(Y))` (voir ci-dessous)

Particularités de notations

- `M(P==1)` peut s'écrire plus simplement `P==1`
- `AG( X => AF(Y))` s'écrit `X --> Y`

Debug

Le model-checker ne peut pas se tromper (les algorithmes sont exacts)

Qu'est-ce qui ne va pas alors ?

- le système est mal modélisé et donc la propriété est bien fausse, il faut utiliser le contre-exemple pour comprendre et corriger
- la propriété n'est pas correctement exprimée, vous pensez qu'elle veut dire quelque chose mais vous vous trompez (ce n'est pas toujours évident de bien formuler)
- le système n'est pas borné (présence infinie de token d'entrée, transition sans place d'entrée etc.)
- le système est trop gros et les algos exponentiels n'aboutissent pas

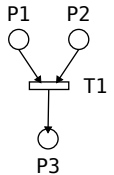
Tutoriel et projet

Tutoriel et début du projet par 2

Propriétés sur les transitions

Réfléchissons

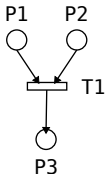
Il n'est pas directement possible de formuler des propriétés sur les transitions en CTL.



Par groupe de 2 réfléchir à comment exprimer (avec GMEC et CTL) que

"la transition T n'est jamais franchissable"

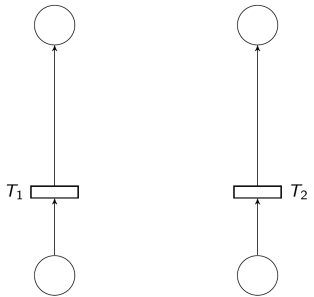
Solution



La transition n'est pas franchissable si nous n'avons jamais ses places d'entrée qui contiennent en même temps un token

```
AG(not M(P1)==1 and M(P2)==1)
```

Plus difficile

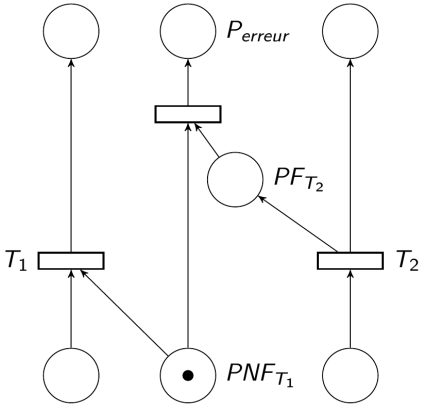


Par groupe de 2 réfléchir à comment exprimer (avec GMEC et CTL) que

"la transition T2 n'est jamais franchie avant T1"

Solution

Mettre en place un sous-réseau observateur




```
AG(M(Perreur) == 0)
```

Modélisation - Les arcs spéciaux

- Arc lecteur
  - arc entrant dans une transition
  - les jetons ne sont pas consommés dans la place entrante quand la transition est franchie
- Arc inhibiteur
  - arc entrant dans une transition
  - la transition peut être franchie seulement si il n'y a pas de jetons dans la place entrante
- Arc réinitialisateur
  - arc sortant d'une transition
  - lorsque la transition est franchie le nombre de jetons de la place de sortie passe à zéro

Suite projet

Suite du projet